Our Case No. 10519/34
(MD-61)

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
APPLICATION FOR UNITED STATES LETTERS PATENT

INVENTORS: Christopher S. Moore
J. James Tringali
Roger W. March
James E. Schneider
Derek J. Bosch
Daniel C. Steere

TITLE: Method for Reading Data in a Write-Once Memory Device Using a Write-Many File System

ATTORNEY: Joseph F. Hetz
BRINKS HOFER GILSON & LIONE
P.O. BOX 10395
CHICAGO, ILLINOIS 60610
(312) 321-4719

Method for Reading Data in a Write-Once Memory Device Using a Write-Many File System

Related Applications

This application claims the benefit of U.S. Provisional Application No. 60/282,723, filed April 9, 2001, and U.S. Provisional Application No. 60/282,790, filed April 9, 2001, both of which are hereby incorporated by reference.

Background

Non-volatile memory is becoming standard in consumer appliances such as digital cameras, digital audio players, and personal digital assistants. The demands for affordable non-volatile storage media coupled with the low cost per bit of information achievable in creating write-once storage elements have given rise to the proliferation of write-once memory devices in the consumer marketplace. Write-many file systems are typically not used to store data in write-once memory devices because these file systems are designed to re-write bookkeeping data structures stored in a given set of memory cells whenever data is added or deleted. Instead, specially-designed write-once file systems, such as ISO9660 and Universal Disk Format (UDF), are used. Unfortunately, data stored using a write-once file system typically cannot be read by a write-many file system. This interoperability problem can affect a large number of users since the prevalence of the MS-DOS FAT file system (ANSI/ISO/IEC 9293-1994) in the personal computer space has lead to the broad acceptance of this write-many file system in the nascent consumer appliance market as well.

There is a need, therefore, for a method for reading data in a write-once memory device using a write-many file system.

Summary

The present invention is defined by the following claims, and nothing in this section should be taken as a limitation on those claims.

By way of introduction, the preferred embodiments described below provide a method for reading data in a write-once memory device using a write-many file system.

In one preferred embodiment, data traffic between a data storage device and a write-once memory device is redirected so that file system structures of a write-many file system do not overwrite previously-stored file system structures. Data traffic between the write-once storage device and a data reading device is also redirected so that a current file system structure of the write-many file system is provided to the data reading device instead of an out-of-date file system structure. In another preferred embodiment, a non-volatile write-many memory array is provided in the write-once memory device to store file system structures of a write-many file system. With these preferred embodiments, data stored during multiple sessions using a write-once file system can be read by a write-many file system, thereby increasing the memory device's interoperability among existing data reading devices. Other preferred embodiments are provided, and each of the preferred embodiments described herein can be used alone or in combination with one another.

The preferred embodiments will now be described with reference to the attached drawings.

Brief Description of the Drawings

Figure 1 is an illustration of a write-once memory device of a preferred embodiment.

Figure 2 is an illustration of a write-once memory device of a preferred embodiment releasably coupled with a data storage device.

Figure 3 is a flow chart of a method of a preferred embodiment for reading data in a write-once memory device using a write-many file system.

Figure 4 is an illustration of a logical organization of a memory array after data from a first session is stored in a write-once memory device of a preferred embodiment.

Figure 5 is an illustration of a write-once memory device of a preferred embodiment releasably coupled with a data reading device.

Figure 6 is an illustration of a logical organization of a memory array after data from a second session is stored in a write-once memory device of a preferred embodiment.

Figure 7 is a first illustration of a master boot record pointer table of a preferred embodiment.

Figure 8 is a second illustration of a master boot record pointer table of a preferred embodiment.

Figure 9 is a third illustration of a master boot record pointer table of a preferred embodiment.

Figure 10 is an illustration of a memory array of a preferred embodiment storing a write state and an address of an alternate data location.

Figure 11 is an illustration of a memory array of a preferred embodiment storing a write state and an address of an alternate data location where a master boot record from a first session is stored.

Figure 12 is an illustration of the memory array of Figure 11 after a master boot record from a second session is stored in the memory array.

Figure 13 is an illustration of the memory array of Figure 11 after a master boot record from a third session is stored in the memory array.

Figures 14 and 15 are illustrations of a memory array of a preferred embodiment where pointers to FATs and a root directory are stored.

Figures 16 and 17 are illustrations of a memory array of a preferred embodiment where a fixed offset is used to locate stored file system structures.

Figure 18 is an illustration of a preferred embodiment where write-many memory is provided in a write-once memory device to store file system structures of a write-many file system.

Detailed Description of the Presently Preferred Embodiments

Turning now to the drawings, Figure 1 is an illustration of a write-once memory device 10 of a preferred embodiment. As shown in Figure 1, the write-once memory device 10 includes a memory array 20 comprising a plurality of write-once field-programmable memory cells. The write-once memory device 10 also includes a memory

array controller 30 that controls read and write operations to and from the memory array 20. The controller 30 can be entirely or partly formed as an array of CMOS gates on the same substrate that supports the memory array 20. The write-once memory device 10 also includes a register 40, which can be separate from the controller 30 (as shown in Figure 1) or integrated with the controller 30.

A field-programmable memory cell is a memory cell that is fabricated in an initial, un-programmed digital state and can be switched to an alternative, programmed digital state at a time after fabrication of the memory cell. For example, the original, un-programmed digital state can be identified as the Logic 0 state, and the programmed digital state can be identified as the Logic 1 state. Because the memory cells are write-once, an original, un-programmed digital state of a memory cell (*e.g.*, the Logic 0 state) cannot be restored once switched to a programmed digital state (*e.g.*, the Logic 1 state). While a conventional two-dimensional memory array can be used, it is preferred that the memory array 20 be a three-dimensional memory array, such as those described in the following patent documents, each of which is hereby incorporated by reference: U.S. Patent Nos. 6,034,882 and 5,835,396 and U.S. Patent Applications Serial Numbers 09/638,428; 09/638,334; 09/727,229; 09/638,439; 09/638,427; 09/638,334; 09/560,626; and 09/662,953. As discussed in those documents, three-dimensional memory arrays provide important economies in terms of reduced size and associated reductions in manufacturing cost. While this write-once memory device 10 takes the form of a solid-state memory device (*i.e.*, a memory device that responds to electrical read and write signals to cause digital information to be read from and stored in a memory array of the device), other types of memory devices can be used, such as optical memory devices (*e.g.*, CD ROMs). Accordingly, the claims should not be read as requiring a specific type of write-once memory device (*e.g.*, solid-state or optical) or specific type of memory array (*e.g.*, two dimensional or three-dimensional) unless explicitly recited therein.

In this preferred embodiment, the memory device 10 takes the form of a compact, handheld unit, such as a memory card or stick. The memory device 10 further comprises an external electrical connector (not shown) and is modular in the sense that it can be easily connected to and disconnected from a device having a mating electrical connector.

For example, as shown in Figure 2, the memory device 10 can be connected to a data storage device 50. As used herein, the terms "connected to" and "coupled with" are intended broadly to cover elements that are connected to or coupled with one another either directly or indirectly through one or more intervening components. Also as used herein, the term "data storage device" (or "authoring device") is intended to broadly refer to any device that can field-program digital data into the memory array 20 of the memory device 10. The data storage device can itself contain or create the digital data or can retrieve the digital data from another location. Examples of data storage devices include, but are not limited to, consumer devices such as a digital audio player, a digital audio book, an electronic book, a digital camera, a game player, a general-purpose computer, a personal digital assistant, a portable telephone, a printer, and a projector. "Digital data" can take any form, such as digital music, digital audio (*e.g.*, voice/speech), digital video, at least one digital still image, a sequence of digital images, digital books, digital text (*e.g.*, news or driving directions), a digital map, games, software, or any combination thereof. It should be noted that a data storage device can also read digital data stored on the write-once memory device 10. For example, a digital camera can both store digital images acquired by the camera and read digital images stored in the write-once memory device 10 for display on the camera's liquid crystal display.

After the data storage device stores digital data in the write-once memory device 10, the memory device 10 can be disconnected from the data storage device 50 and connected to a data reading device 60. The term "data reading device" (or "reading device") is intended to broadly refer to any device that can read digital data stored in the memory array 20 of the memory device 10. Examples of data reading devices include, but are not limited to, consumer devices such as a digital audio player, a digital audio book, an electronic book, a digital camera, a game player, a general-purpose computer, a personal digital assistant, a portable telephone, a printer, and a projector. In addition to reading data, a data reading device can also write data in the memory device 10.

As described in the background section above, because write-many file systems require a memory device to be re-writeable, write-once file systems have been developed to store data in a write-once memory device. In most applications, the write-once file

system must be used to both write data to and read data from the memory device. The assignee of the present application has presented an alternative to this all-or-nothing approach in U.S. Patent Application Serial No. 09/748,589, which is hereby incorporated by reference. With this solution, even though a write-once file system is used to write data to the memory device, a write-many file system, such as the ubiquitous DOS FAT file system, can be used to read data from the memory device. This allows a write-once memory device, which is not FAT compatible for write operations, to be FAT compatible for read operations.

As described in U.S. Patent Application Serial No. 09/748,589, memory cells are reserved in the appropriate locations in the memory device for file system structures of a write-many file system. After data is stored in the write-once memory device using a write-once file system, file system structures for both the write-once file system and a write-many file system are stored so that the stored data can be read by any data reading device using the write-many file system or the write-once file system. The writing of file system structures is sometimes referred to as "closing" the memory device. Because the reserved memory cells can only be written into once, write-many file system structures cannot be updated when data is added to the memory device. Since data stored after the memory device has been "closed" will not be reflected in the previously-written file system structures, the newly-stored data will not be readable by a data reading device using a write-many file system. Accordingly, file system structures are preferably stored when the memory device is filled or when a user indicates that no more data is to be stored in the memory device. This approach is perfectly suitable for many applications. For example, a user accustomed to using a conventional film-based camera may prefer to "close" a write-once memory device only after the memory device has been filled with pictures. However, in some applications, a user may wish to store data on the write-once memory device over multiple sessions. The following preferred embodiments find use in those applications.

By way of overview, the preferred embodiments described below allow a memory device to be "closed" multiple times (*e.g.*, after each session), thereby allowing a write-many file system to read data stored after the memory device was first "closed." In this

way, data stored in the memory device over multiple session using a write-once file system can be read by a write-many file system. As used herein, a "session" includes the time period when data is stored in the memory device and ends when file system structures associated with that data are stored in the memory device. Thus, a new session takes place when data is stored in the memory device after file system structures from a previous session have been written in the memory device. A session can end in response to a user action (*e.g.*, when a user ejects the memory device 10 from a data storage device 50) or can end when the memory device is completely or nearly completely filled with data. While a memory device can be removed from a data storage device between sessions, multiple sessions can take place without removing the memory device from the data storage device. This can occur, for example, when a user presses a button on a digital camera to indicate that he has ended a session and then decides to take additional pictures before removing the memory device.

This preferred embodiment will be now be illustrated with reference to the DOS FAT file system. This preferred embodiment allows a write-once memory device to be "closed" multiple times, thereby allowing data stored over multiple sessions to be read by a DOS FAT file system. In this way, the write-once memory device is both updateable and DOS FAT read compatible. During formatting of the memory device 10, one or more partitions are created, and a master boot record (MBR) is written at address zero, per DOS FAT protocol. The MBR points to the location of partition boot records (PBRs) of the memory device. A PBR is found at the start of each partition of the memory device and contains information that can be used to locate the partition's file allocation table (FAT) and root directory. A FAT is a table structure that stores a chain of blocks in use by a file. The end of the chain of blocks is indicated with an end-of-file marker. A FAT also stores information on which blocks are free and which blocks are bad. Two copies of the FAT are typically stored in a partition. A root directory links descriptive information about a file (*e.g.*, its name, size, attributes, etc.) with information stored in the FAT about its physical location in the memory device. Preferably, the memory locations where the DOS FAT file system expects to find file system structures are reserved locations to prevent user data from being stored therein. Since the PBR, FATs,

and root directory can be derived from the MBR, a write-many file system only needs to know the location of the MBR to determine the location of the other file system structures, as described below.

Turning again to the drawings, Figure 3 is a flow chart of a method of a preferred embodiment for reading data in the write-once memory device 10 using a write-many file system. This method will be described in conjunction with the illustrations of the memory array 20 in Figures 4 and 6. The memory array 20 in Figures 4 and 6 has only a single partition to simplify this example. It should be noted that more than one partition can be used. Figures 4 and 6 also diagrammatically show the interrelation between the MBR, PBR, FATs, and root directory. Returning to the flow chart, the memory device 10 is first connected with a data storage device 50 (act 100). Next, data 200 is stored in the write-once memory device 10 during a first session (act 110). In operation, the digital storage device 50 sends data to the controller 30 of the memory device 10, which stores the data in the memory array 20. At the conclusion of the first session, a first set of file system structures 210 for the write-many file system is stored in the memory array 20 (act 120) (*i.e.*, the memory device 10 is "closed" for the first time). As used herein, a "set" can include one or more than one element. Here, the PBR, FATs and the root directory are written into the memory cells reserved for these file system structures. If the master boot record (MBR) is not stored during formatting of the memory device 10, it can be stored with the other file system structures. (As described in more detail below, the data storage device 50 also stores a pointer to the location of the MBR. Since the current version of the MBR is located at address 0 here, a pointer does not need to be stored. If the MBR were initially stored at another location, the technique described below could be used to locate the MBR.) A set of file system structures 220 for the write-once file system is also stored (act 130) to allow the data 200 to be read by a data reading device using the write-once file system.

With the first set of file system structures 210 stored, the memory device 10 can be removed from the data storage device 50 and inserted into a data reading device 60 using the DOS FAT file system (see Figure 5). The data reading device 60 sends a read address zero command to the memory device 10 per DOS FAT protocol, and the

controller 30 returns the MBR stored in address 0. (As described in more detail below, the controller 30 does not always return the MBR stored in address 0 in response to a command to read address 0. However, at this time, the current version of the MBR is stored at address 0.) Because the location of the PBR can be derived from the MBR and the locations of the FATs and the root directory can be derived from the PBR, the write-many file system of the data reading device 60 can read any of the data stored in the memory device 10 during the first session even though a write-once file system was used to store the data.

With reference to Figure 6, the memory device 10 is later re-inserted into the data storage device 50 (or inserted into a different data storage device), and data 230 from this second session is stored in the memory device (act 140). At the conclusion of the second session, a second set of file system structures 240 for the DOS FAT file system is stored in the memory array 20 (act 150). As shown in Figure 6, the second set of file system structures 240 includes a new MBR, PBR, FATs and root directory, which include information on the data stored during both the first and second sessions. Because the memory array 20 comprises write-once memory cells, the first set of file system structures 210 cannot be updated to reflect the data stored during the second session. Accordingly, the second set of file system structures 240 is stored in an available location in the memory array. In one embodiment, the controller 30 of the memory device 10 redirects the second set of file system structures 240 to an available location in the memory array 20 to avoid overwriting the first set of file system structures 210. A new set of file system structures 220 for the write-once file system is also stored at the conclusion of the second session (act 160).

Because a DOS FAT file system always looks for file system structures at the same location of the memory array 20 (the locations occupied by the first set of file system structures 210), the DOS FAT file system will not read the second set of file system structures 240. Accordingly, the DOS FAT file system will not be able to read the data stored during the second session 230. However, if data traffic between the memory device 10 and the data reading device 60 were redirected such that second set of file system structures 240 (instead of the first set of file system structures 210) were provided

to the data reading device 60, the data reading device 60 would have access to data stored during both the first and second sessions. To achieve this result, a pointer to the memory location storing the up-to-date MBR can be stored in the memory device 10. When a write-many file system attempts to read a memory location containing an out-of-date file system structure, the current file system structure stored at the memory location indicated by the pointer will be returned.

Since a new pointer would be written every time the bookkeeping data is updated (*i.e.*, with each "close" of the memory device 10), it is preferred that a portion 260 of the memory array 20 be reserved for pointers to avoid the possibility of running out of memory space to store the pointers. Preferably, this portion is outside of the partition so that user data cannot be written into the MBR pointer portion 260 and so that pointers cannot be "obliterated" by the user. Alternatively, the portion 260 can be stored as a reserved or read-only file within the partition that can be written into only when the controller 30 is in a special mode. Of course, other alternatives are possible. It is also preferred that a pointer be written after (not before) data is stored in the memory array 20. In this way, if a power loss occurs during a write operation, the last-stored pointer would point to the previously-written MBR instead of an empty (or partially written) set of memory cells.

Figures 7-9 are illustrations of the memory cells of the MBR pointer portion 260. In these figures, a line of four bits is shown to simplify the drawings, and each line of written memory cells corresponds to a pointer. Figure 7 shows the initial state of the memory cells in the MBR pointer portion 260. As discussed above, in this embodiment, the MBR written to the memory array 10 during formatting is accurate for the data stored during the first session. Accordingly, a pointer is not written into the MBR pointer portion 260 of the memory array 20 at the conclusion of the first session. If a new MBR is stored at address 0101 at the end of the second session, a corresponding pointer is stored in the MBR pointer portion 260, as shown in Figure 8. Similarly, if a new MBR is stored at address 1101 at the end of a third session, a pointer to that address is stored in the MBR pointer portion 260 following the previously-stored pointer, as shown in Figure 9.

When the memory device 10 is connected to a data reading device 60, the memory device 10 is powered-up, and circuitry in the controller 30 queries the MBR pointer portion 260 to find the last valid entry. (This can also occur in response to a commanded device reset operation.) To find the current pointer, the controller 30 preferably identifies the last line of written data by locating the first line of unwritten memory cells. For example, the controller 30 can perform the following sequence:

```
if (*overwrite_buf != UNPROGRAMMED) {
  while (*overwrite_buf != UNPROGRAMMED) {
    *MBR = *overwrite_buf++;
  }
}
```

This and other temporal-to-spatial mapping techniques are described in U.S. Patent Application Serial No. 09/748,589.

The controller 30 then stores the current pointer in the memory device's register 40 (*e.g.*, the memory device's RAM). This stored address will be used when any device sends a request to read address zero — the standard location of the MBR in the DOS FAT file system. In this way, all subsequent requests for the MBR are redirected to the location indicated by the pointer. For example, when the data reading device 60 sends a read address zero command to the memory device 10, the controller 30 returns the new MBR located at the address stored in register 40 instead of the out-of-date MBR stored at address zero. Accordingly, the most up-to-date version of the MBR is sent to the data reading device 60 even though it was not stored at the address expected by the data reading device's DOS FAT file system (logic address zero). As discussed above, the location of the current PBR is derived from the MBR and the locations of the current FATs and root directory are derived from the current PBR. Accordingly, the data reading device 60 can read the data stored during both the first and second sessions.

It should be noted that the functionality described above can be distributed among the data storage device 50, the memory device 10, and the data reading device 60 in any desired manner. In one preferred embodiment, the functionality resides solely in the

memory device 10. For example, the controller 30 of the memory device 10 can direct the storage of an MBR to an available location in the memory array 20 and store a pointer to the MBR after it is written. The controller 30 can also retrieve the pointer during power-up of the memory device 10 and be responsible for returning the MBR stored in the memory location indicated by the pointer in response to a read address zero command from the data reading device 60. With this implementation, any special knowledge of write-many emulation would be limited to the write-once memory device 10 itself, and no hardware or software adjustments would be needed in either the data storage device 50 or the data reading device 60. To lower the cost and complexity of the memory device 10, some of the emulation tasks can be distributed between the memory device 10 and the hardware and/or software of the data storage device 50. For example, the data storage device 50 (instead of the controller 30) can be responsible for storing the MBR in an available location in the memory array 20 and storing a pointer to the MBR in the reserved portion 260. A file system copy-and-invalidate resource of the data storage device 50 can also keep track of over-write bookkeeping. With either of these implementations, no modification is needed to the data reading device 60, thereby providing a high degree of interoperability with all existing playback devices. In addition, the process of redirecting the request for the MBR is completely hidden from the data reading device 60. However, if desired, some of the emulation tasks can be distributed to the data reading device 60 as well. For example, hardware and/or software of the data reading device 60 (instead of the controller 30 of the memory device 10) can be responsible for retrieving the stored pointer and sending a read request designating the address indicated by the stored pointer instead of address zero.

In another preferred embodiment for achieving rewrite emulation using a write-once storage medium, redirection of overwritten data is determined at the level of the smallest grouping of data allowed by the memory device's buffer allocation scheme. Typically, this is referred to as a sector of data based on the nomenclature used in conventional disk-based systems. In this embodiment, the memory array of the write-once storage device is constructed in such a fashion to allow two extra fields of information to be stored in association with an amount of stored user data. As shown in

the memory array of Figure 10, the two extra fields indicate a write state 500 and an address of an alternate data location 510. When a write buffer passes data to the write-once memory device, an address chain sequencer 520, which can be implemented in the memory device, queries the associated write state field 500. The write state field can include one of the states set forth in Table 1 below.

State	Description
UnWritten	Indicates that this section of the device is able to accept data.
Written	Indicates that this section of the device is holding valid written data.
OverWritten	Indicates that this section of the device has been overwritten and that the new valid data lies at the address indicated by the NxtAddr field.

Table 1: WriteState Enumeration

The initial state of the WriteState sideband fields is UnWritten. When a block of data is sent to a sector address in the UnWritten state, the data is allowed to be stored at that address, and the WriteState sideband field is transitioned to the Written state. The next write to the same sector address (now in a Written state) will cause that sector to become invalidated and the new data to be redirected to a sector in the UnWritten state. The transition from Written to OverWritten involves several related actions. The controller 30 finds the next sector marked UnWritten and deposits the sector data at that location. The WriteState sideband field of the alternate location is then updated to Written. In addition, the address of this alternate location is placed in the NxtAddr sideband field of the detected overwrite, and its WriteState sideband field is simultaneously be updated to OverWritten. In this manor, multiple overwrites of a given address will form a chain of addresses terminating in the last data update received. In order to mitigate the issues resulting in software updates to addresses already taken by the controller 30 for re-direction, all candidates for over-write sectors are preferably kept in an area of memory inaccessible to the external device supplying data. As with the other preferred embodiments described above, the functionality described with respect to this

preferred embodiment can be distributed among the data storage device 50, the memory device 10, and the data reading device 60 in any desired manner.

Figure 11-13 will be used to illustrate the operation of this preferred embodiment when a DOS FAT file system is used. In Figure 11, an MBR from a first session is written at address 0, per DOS FAT protocol. Because valid data is written into this sector, the WriteState field is "Written," and the NxtAddr field contains no data. When an MBR from a second session is to be stored in the memory array, the controller 30 prevents this new MBR from overwriting the data in address zero and stores the MBR in an available location in the memory array. As shown in Figure 12, the MBR from the second session is stored at address 100. The controller 30 also changes the WriteState field of address zero to "OverWritten" and writes address 100 into the NxtAddr field. When the memory device is read by a data reading device, a read address zero command will be sent to the controller 30. In response to this command, the controller 30 examines address zero and finds that the data stored therein is designated as overwritten and that current, valid data is stored at address 100. The controller 30 then returns the MBR from the second session stored at address 100 to the data reading device instead of the out-of-date MBR from the first session stored at address 0. As shown in Figure 13, when a MBR from a third session is to be stored in the memory array, the controller 30 stores the new MBR in an available location in the memory array (here, address 99). The controller 30 also changes the WriteState field of address 100 to "OverWritten" and writes address 99 into the NxtAddr field. In response to a read address zero command from a data reading device, the controller 30 examines address zero and finds that the data stored therein is designated as overwritten and that current, valid data is stored at address 100. When the controller examines address 100, it finds that the data stored therein is also designated as overwritten and is directed to address 99. As shown by this example, the controller "walks the chain" from address 0 to address 100 to address 99 to find the current valid MBR.

In the above examples, the data reading device 60 sent a read address zero command to the memory device 10 to read the MBR. This command was translated either by the memory device 10 or by hardware and/or software in the reading device 60

so that the current MBR, which was not stored at address zero, was returned to the data reading device 60. The data reading device 60 then derived the location of the current PBR from the current MBR and, from the current PBR, derived the locations of the current FATs and the current root directory. However, some data reading devices send a request to read the address where they expect to find the PBR (not the MBR) so that resources are not spent deriving the PBR from the MBR (thereby increasing performance). The preferred embodiments described above can be modified so that the pointer indicates the location of the current PBR instead of the location of the MBR. As discussed above, the location of the current FATs and the current root directory can be derived from the current PBR stored in the location indicated by the pointer.

In the embodiments described above, a new PBR, new FATs and a new root directory were written to the memory device 10 when data was added or obliterated. (The "obliterate" function, which makes previously-stored data difficult or impossible to read by overwriting at least a portion of the stored data with a destructive pattern, is described in U.S. Patent Application Serial No. 09/638,439, which is assigned to the assignee of the present invention and is hereby incorporated by reference.) In another preferred embodiment, both the MBR and the PBR remain static, and only new FATs and a new root directory are written. In this embodiment, pointers to the current FATs and root directory are stored instead of a pointer to the MBR (or to the PBR). This alternative will be discussed in conjunction with the illustrations of the memory array 20 in Figures 14 and 15. It should be noted that the addresses in this example and in the other examples provided herein are used for simplicity and are not meant to imply an actual address used by a DOS FAT file system. In Figure 14, data from a first session is stored at address 00100, and the FATs and the root directory associated with this data are stored at addresses 00010 and 00011, respectively. Hardware and/or software in the data storage device 50 (and/or the controller 30 in the memory device 10) stores pointers to the FATs and the root directory in a pointer table outside of the partition. As shown in Figure 14, these pointers are stored in the first two lines of the table.

Turning now to Figure 15, data from a second session is stored at address 00101. In this example, the MBR and the PBR stored at 00000 and 00001 are written when the

memory device 10 is formatted, and new MBRs and PBRs are not stored in the memory device 10 after data from the second session is stored. Instead, only new FATs and a new root directory are stored. Pointers to address 00110 and 00111 (the locations of the new FATs and the new root directory) are stored in the next two available lines in the pointer table following the previously-written pointers. If the root directory immediately follows the FATs (or vice versa) and the size of the FATs and root directory is known (*e.g.*, from the PBR), a single pointer can be stored instead of two pointers.

When the memory device 10 is powered-up, the controller 30 uses the temporal-to-spatial mapping technique described above to identified the pointers to the FATs and the root directory and stores their addresses in the register 40. For example, controller 30 can locate the first line of unwritten memory cells and identify the previous line as the address of the root directory and the line before that as the address of the FATs. From the PBR, the controller 30 also identifies the addresses where the data reading device 60 would expect to find the FATs and the root directory (address 00010 and 00011). Alternatively, if the FATs and root directory are stored at predetermined addresses in the memory array 20, the controller 30 can be pre-programmed with those address. When the data reading device 60, sends a command to read address 00010 to read the FATs, the controller 30 returns the current version of the FATs stored at 00110 instead of the out-of-date FATs stored at 00010. A similar operation occurs with respect to the root directory.

In the preferred embodiments described above, a pointer to one or more current file system structures was stored in a table or in a field associated with a set of memory cells for storing data (*e.g.*, a sector of data). It should be noted that the pointer(s) can be stored in any other suitable location. Additionally, data can be redirected without the use of a stored pointer. The preceding example where the root directory immediately follows the FATs and the size of the FATs and root directory is known will be used to illustrate this alternative along with Figures 16 and 17. Figure 16 shows data from a first session stored in a partition of the memory array 20 and the FATs and the root directory associated with this data stored in an area outside of the first partition in the memory array. In this embodiment, the FATs and the root directory are stored at the bottom of

this area (the "file system structure portion"). The FATs and the root directory can also be stored in an area within the partition that is unavailable to store user data. Figure 17 shows data from a second session stored in the partition and the FATs and the root directory associated with this data stored in the next available memory locations preceding the previously-stored FATs and root directory. When a data reading device 60 sends a command to the memory device 10 to read the addresses that it expects to find the FATs and the root directory (*e.g.*, based on the PBR), the controller 30 locates the last line of unwritten memory cells in the area outside of the partition. Because the FATs and the root directory are of a fixed, known size, the controller reads the following X lines, where X is a fixed offset determined by the size of the FATs and the root directory. The FATs and the root directory stored in those lines are then returned to the data reading device 60.

In the preferred embodiments described above, the file system structures for the write-many file system were stored in a memory array of write-once memory cells. In another embodiment, which is shown in Figure 18, a write-once memory device 300 is provided with both a write-once memory array 310 and a non-volatile write-many memory array 320. Although shown as a separate component, the write-many memory array 320 can be integrated with the write-once memory array 310 or with the controller 330. The write-many memory array 320 can take the form, for example, of an electrically-erasable programmable read-only memory, such as Flash memory. In operation, after data is stored in the write-once memory array 310, hardware and/or software in a data storage device (or the controller 330 in the write-once memory device 300) would stored file system structures of a write-many file system in the write-many memory array 320. Because these structures are written in re-writable memory, they can be updated when additional data is stored in the write-once memory array 310. For example, new FATs and a new root directory can be re-written over the previously-written FATs and root directory. When a data reading device sends a read command for the addresses storing these file system structures, the controller 330 reads those addresses from the write-many memory array 320 instead of the write-once memory array 310. Alternatively, hardware and/or software in the data reading device can send a command

to read the addresses from the write-many memory array 320 instead of the write-once memory array 310.

For simplicity, the operation of the preferred embodiments described above was illustrated in conjunction with the DOS FAT file system. It should be noted the techniques described above for achieving rewrite emulation using a write-once memory device are not limited to the DOS FAT file system and that any other write-many file system can be used. Additionally, in the above-described preferred embodiments, the pointer only pointed to the MBR. In an alternate embodiment, a pointer to the PBR, FATs, and/or root directory can also be used in addition to or instead of the pointer to the MBR. Also, while the preferred embodiments discussed above stored file system structures for both a write-many and a write-once file system, file system structures of two different write-once file systems or two different write-many file systems can be used. Further, while the preferred embodiments were discussed in terms of two file systems, the preferred embodiments can be extended to achieve interoperability among three or more different file systems. In yet another alternate embodiment, only write-many file system structures are stored (and not write-once file system structures) even though a write-once file system is used to store data and write-many file structures.

It is intended that the foregoing detailed description be understood as an illustration of selected forms that the invention can take and not as a definition of the invention. It is only the following claims, including all equivalents, that are intended to define the scope of this invention. Finally, it should be noted that any aspect of any of the preferred embodiments described herein can be used alone or in combination with one another.